Toward Dependable Embedded Model Predictive Control

Tor A. Johansen

***Abstract*—While model predictive control (MPC) is the industrially preferred method for advanced control in the process industries, it has not found much use in consumer products and safety-critical embedded systems applications in industries such as automotive, aerospace, medical, and robotics. The main barriers are implementability and dependability, where important factors are implementation of advanced numerical optimization algorithms on resource-limited embedded computing platforms and the associated complexity of verification. This challenge comes from a requirement of the use of ultrareliable hardware and software architectures in safety-critical applications, low-cost hardware in consumer products, or both. This paper surveys the state-of-the-art in the emerging field of dependable embedded MPC, and discusses some key challenges related to its design, implementation, and verification. A novel result is the study of a simulator-based performance monitoring and control selection method that monitors and predicts MPC performance and switches to a highly reliable backup controller in cases when the MPC experiences performance issues.**

***Index Terms*—Dependability, embedded systems, fault-tolerance, numeric optimization, system safety.**

I. Introduction

MODEL predictive control (MPC) was developed as a practical implementation of optimal feedback control for multivariable processes that were subject to input and output constraints. At the current time t_0, when the system is in the state defined by the vector $x(t_0)$, MPC solves the following optimization problem in order to compute optimal control inputs $u(t_0), \ldots, u(t_N)$ on the time horizon N:

$$\min_{u(\cdot),s(\cdot)} J(u(t_0),\ldots,u(t_N),s(t_0),\ldots,s(t_N);x(t_0)) \text{ subject to}$$

$$x(t_{k+1}) = f\left(x(t_k),u(t_k),t_k\right),\ x(t_0) \text{ given}$$

$$g\left(x(t_k),u(t_k),s(t_k),t_k\right) \leq 0, \quad k = 0,1,2,\ldots N$$

where J is a cost-function, f a discrete-time dynamic model, g is a function that represents constraints, and vector $s(t_k)$ contains slack variables that are used to relax the constraints to guarantee the existence of a solution. At every time instant, $t_1, t_2, \ldots$, the procedure is repeated based on updated information. MPC is proven to enable significant performance improvements in a range of applications and has been particularly successful in the process and petroleum industries [1]. The industrial success of MPC has led to intense research, and theoretical issues such as stability and optimality, and how they are influenced by real-time computational resource limitations, are by now fairly well understood [2].

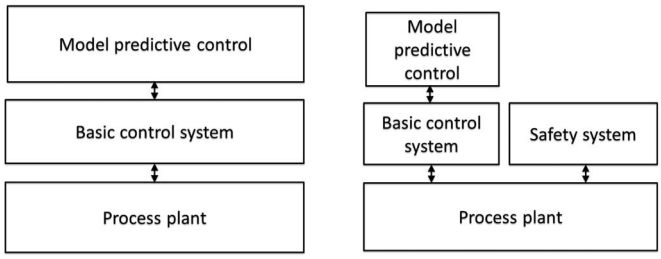

Fig. 1. Two common variations of conventional MPC. Left: (a) Conventional MPC system architecture. Right: (b) Conventional MPC system architecture with independent safety system.

The implementation of MPC in the process and petroleum industries has typically been as a high-level multivariable controller at a level above a layer of basic single-loop control, as illustrated in Fig. 1. The basic control has typically been responsible for plant stabilization, disturbance rejection, and set-point tracking. Thus, the interface between the basic control and MPC is typically set-point commands to low-level PID-type controllers. The basic control is typically implemented within industrial programmable logic controllers (PLCs) or industrial distributed control systems (DCSs) that are based on highly dependable hardware and firmware. However, although the basic control system provides a significant degree of both control performance and safety by itself, it is sometimes accompanied by an independent safety system. The safety system is designed to take the plant to a safe state when unsafe conditions are detected or predicted, or call for intervention or manual control by an operator. This often involves reconfiguring, isolating, or shutting down a part of the plant. Although not explicitly represented in the figure, a typical architecture may also include a higher level optimization of steady-state conditions. This often uses nonlinear static models in order to provide reference points for inputs and states to the MPC and is referred to as real-time optimization. Alternatively, the static and dynamic optimization is combined in so-called economic MPC, e.g., [3].

With the basic control performance and safety already provided, it has been convenient to allow the MPC to be implemented

Manuscript received July 4, 2014; revised August 14, 2014 and September 30, 2014; accepted November 2, 2014. This work was supported by the Research Council of Norway, Statoil and DNV through the Center of Excellence of Autonomous Marine Operations (AMOS) and Systems, by the Research Council of Norway and Statoil through the PETROMAKS project Enabling High-Performance Safety-Critical Offshore and Subsea Automatic Control Systems using Embedded Optimization, and by the European Commission through the Marie Curie ITN Training in Embedded Model Predictive Control.

The author is with the Center for Autonomous Marine Operations and Systems, Department of Engineering Cybernetics, Norwegian University of Science and Technology, 7491 Trondheim, Norway.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSYST.2014.2368129

in computationally powerful and inexpensive, however, not very reliable off-the-shelf computer hardware and software. This usually means PC-type computers running office/server-type of operating systems not designed with the strict resource control and scheduling needed for real-time response guarantees and ultrahigh reliability. Moreover, the numerical optimization required for MPC has conveniently been based on sophisticated off-the-shelf numerical code and libraries that have been developed and improved over long time. Often, this code is proprietary and/or binary and therefore to be considered more or less as a "black box" although basic algorithms have known properties. Arguably, the dependability of the standalone MPC module has received only modest attention in many applications and implementations, and perhaps been driven more by availability and reliability requirements rather than safety.

Currently, there is significant research activities on algorithms and software for embedded MPC, aiming to implement MPC either on ultrareliable industrial real-time computer platforms such as PLCs and DCSs, or custom-embedded system hardware (HW) with microcontrollers, field-programmable gate arrays (FPGAs), or application-specific integrated circuits (ASICs). This development is driven by commercial interests in new applications and other industries following the success of conventional MPC in process control.

1) Desire to use MPC in applications that are characterized by relatively fast dynamics where the real-time aspects of MPC computations must be taken very seriously, e.g., subsea petroleum production (e.g., [4] and [5]), oil well drilling control systems (e.g., [6]), robotic systems (e.g., [7]), aerospace (e.g., [8]), electric power generation and distribution systems (e.g., [9]), and scheduling of computing resources, e.g., in cloud computing [10].
2) Desire to use MPC in applications that requires extremely fast sampling and updates, e.g., power electronics [11].
3) Desire to use MPC in products, which require low-cost embedded systems, e.g., consumer electronics, medical devices (e.g., [12]), and cars (e.g., [13]).
4) Desire to use MPC for low-level control, also for plants that are not prestabilized (e.g., [14]).
5) Desire to use MPC in safety-critical applications.

This means that MPC might be used within different embedded system architectures, where some of them will impose significantly stronger responsibility on the MPC in order to not only provide high-level performance-improving control, but also basic and safety-related control functionality such as stabilization, disturbance rejection, and fault tolerance. In safety-critical applications, the consequences of failure of the MPC system may be catastrophic and unacceptable.

The aforementioned trends mean that the embedded MPC may need to be made significantly more dependable than conventional MPC. Dependability means the system's ability to avoid failures with unacceptable consequences for the system's functionality, [15]. While "external" faults in sensors, actuators, and the plant can lead to control system failure with unacceptable consequences, the focus of this study is on the dependability and vulnerability of the MPC computer control system with its computer hardware and software. The limited resources

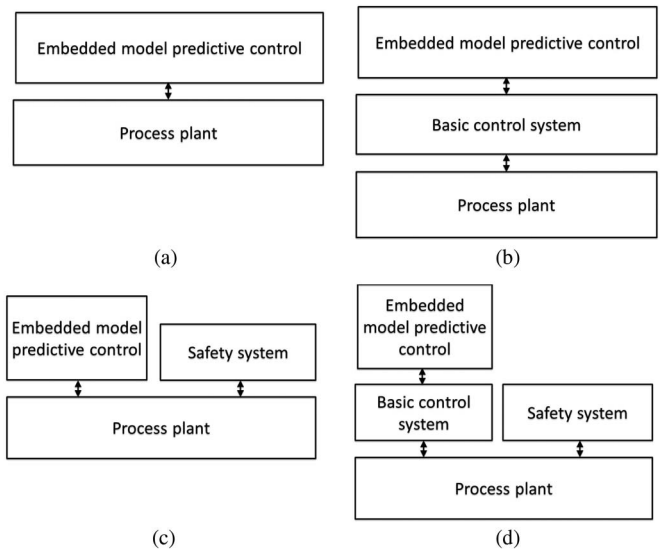

Fig. 2. Four possible embedded MPC system architectures. Top left: (a) "Bare" embedded MPC system architecture. Top right: (b) Embedded MPC system architecture with separate basic control. Bottom left: (c) Embedded MPC system architecture with independent safety system. Bottom right: (d) Embedded MPC system architecture with separate basic control and independent safety system.

of an embedded system (such as power, memory, processing capacity, and software libraries) in combination with requirements for predictable performance poses the main barriers for replacing conventional industrial control algorithms with MPC in embedded systems. For completeness, we remark that in addition to handling of internal faults, the MPC's tolerance to external faults is also an active research area, e.g., [9], [16]–[18].

A summary of potential embedded MPC system architectures is given in Fig. 2. As an extreme case, the architecture in Fig. 2(a) has no additional safety system or even basic control and stabilization, which increases the dependability requirements to the core embedded MPC module. The desire to reduce cost in products where MPC is embedded may favor this architecture, e.g., in automotive applications [13].

In other applications with strong couplings and nonlinearities, a basic control layer based on PID-controllers may limit the achievable performance and therefore be unacceptable, e.g., [4] and [5], favoring the architecture in Fig. 2(c). On the other hand, the architectures in Fig. 2(b) and (d) are similar to conventional MPC and may be favored in applications when the MPC is embedded in an existing fault tolerant system design, e.g., [9].

Many aspects of dependable embedded MPC are currently receiving increasing interest in the literature, whereas others are not receiving the attention that may be needed in order to enable embedded MPC in certain application areas.

The purpose and contribution of this paper is to provide a holistic systems perspective and discuss some requirements and challenges for dependable embedded MPC. Section II surveys embedded MPC architectures, algorithms, hardware, and software. Section III discusses the key aspects of dependability, and together, these two sections lay a foundation for dependable embedded MPC. In Section IV, this leads to some novel results and advice for practitioners on how dependable

embedded MPC architectures, algorithms, and computer hardware and software can be designed, implemented, and verified for demanding and safety-critical control applications. A key element is a fault-tolerant architecture that includes a simulation-based performance monitor to select between different control solutions that is presented and illustrated using a simulation example in Section IV. Section V ends with some concluding remarks on the future of dependable embedded MPC.

II. Embedded MPC Architectures, Algorithms, Hardware, and Software

This section will provide some further discussion on the embedded MPC architectures illustrated in Fig. 2 and the underlying algorithms, hardware, and software required to implement embedded MPC in a highly resource-limited real-time embedded control system.

The limited resources in combination with requirements for predicable performance poses the main barriers for replacing conventional industrial control algorithms with MPC in embedded systems. Limited power and cooling means limited computational resources for computationally intensive numerical optimization with hard real-time constraints that normally rely on the accuracy of floating-point arithmetic. With complex algorithms that have a relatively large footprint, limited computer memory might reduce performance. One reason is that the transfer of data and instructions from the slower memory modules to the central processing elements might be a bottleneck leading to latencies and unused processing capacity. Another reason is that it also means limited opportunities to store precomputed data structures to support the real-time processing. Redundant hardware or functionality is often a highly useful factor in developing dependable systems, but the opportunities are limited by shortage of processing and storage resources.

A. Embedded MPC Architectures

The existence of a dedicated and independent safety system, as shown in Fig. 2(c) and (d), means that the safety of the process is not a primary responsibility of the MPC. Still, even with the existence of such as safety system, there might be very demanding requirements for the MPC since the cost and consequences of activating the safety system might be significant as it may lead to a shutdown causing several hours or days of nonproductive time of an industrial asset. Performance and reliability of the MPC may be a primary concern even if it is not safety critical.

One should have in mind that, in some applications, it might be difficult, or even impossible, to develop an independent safety system that could respond safely to control failures. Reasons for this could be that there is no well-defined fail-safe state in the case of control system failure (think of an aircraft or a ship loosing steering or propulsion). In such cases, the faulty control system should perhaps be overridden by manual (human) control, or some redundant control system (hardware or functional redundancy) should take over. Such redundancy is provided by the separate and independent basic control system in Fig. 2(b). As will be investigated in Section IV, the control reconfiguration required to implement this is quite complicated as it may require a separate diagnosis function in order to identify if the embedded MPC output is to be trusted and used, or not.

The "bare" architecture in Fig. 2(a) provides no safety or functional redundancy beyond what is built into the embedded MPC. There are indeed several safety functions that are recommended to be integrated within an MPC, e.g., state constraints. Since there is no external backup solution, the demands for verification and validation of the embedded MPC hardware and software are likely to be very strict if the consequences of failure are severe. As will be discussed later, this has significant consequences for the design and analysis of the embedded MPC.

The choice of control system architecture is likely to be highly dependent on several factors, such as requirements for dependability, hardware and software technology requirements, and the related costs of design, verification, and validation. As will be introduced in Section IV, there are additional novel architectures beyond those in Fig. 2 that might be attractive. Systematic system design and analysis frameworks are available in the form of industry standards such as IEC 61508 and 61511, and MIL-STD-882C. They provide methods and tools for the design and analysis based on the particular requirements of the application. It is considered highly unlikely that one can find single architectures, algorithms, hardware and software implementations that fit all applications, and a system theoretical safety approach may be fruitful in complex systems [19]. The use of MPC is particularly fruitful in the systems theory approach to safety, as MPC provides a systematic approach to integrate safety constraints with other operational constraints, objectives and performance criteria for reference tracking, control effort use, or economic objectives. The use of a systems theory approach is particularly evident for the design, development, implementation, verification and certification phases, although relevant also for analysis of requirements and feasibility, and specification.

B. Hardware and Software Environments

While today's conventional MPC is typically run under some Windows or Unix/Linux variant on PC/server-type of computers, embedded MPC may run in a variety of different platforms:

1) **Industrial controllers**, typically well-proven processing units with floating-point coprocessors and fairly large amounts of memory for program and data. These standardized controllers are designed to operate in a DCS and typically run some real-time operating system and comes with well-proven standard libraries for communication, networking, input/output, configuration management, control and signal processing. Moreover, they usually support custom programs (such as MPC numerical optimization) using high-level programming languages such as C or C++.
2) **PLCs**, which are similar to, but usually designed to be even more robust than, industrial controllers. They can operate standalone or within a DCS. Usually, they have processors with fairly limited processing and memory capacity, and only top-end models tend to have floating-point coprocessors. The access to resources and

programming is usually limited to specific block-oriented or highly structured programming languages (such as IEC 61131-3) with standard blocks for communication, networking, input/output, configuration management, control and signal processing. Such limitations contribute to less risk for users' programming errors causing system failure. In addition, some offer structured programming environments with C-like language, or reduced C syntax and a few standard libraries, which can be used to implemented embedded MPC [4], [20]. They are designed using extremely robust and reliable electrical and mechanical design and components, and made for high endurance in harsh industrial environment characterized by large temperature ranges, humidity, dust, vibrations, electromagnetic interference, etc.

3) While industrial controllers and PLCs are standardized modules that can work as building blocks in the design of a large industrial system, many embedded systems designed for serial production (e.g., in the automotive and aerospace industries) tend to be custom designed as **electronic control units** with dedicated circuit boards that include input/output and interface electronics in addition to processing units. Embedded MPC can be implemented in the supported programming languages and operating environment, typically ranging from small microcontrollers (without any floating-point coprocessing) [21], to very powerful multicore digital signal processing chips with floating-point capacity, and to FPGAs [22]–[26] and ASICs [27].

C. Embedded MPC Algorithms and Software

The workhorse of MPC software is numerical optimization, usually in the form of quadratic or linear programming solvers (for so-called linear MPC that is characterized by a linear model, linear constraints, and a linear or quadratic cost function) or nonlinear programming solvers (for so-called nonlinear MPC). With a few notable exceptions, the commercial or public domain numerical optimization software available for use in MPC is not open-source, which usually means that one is limited to Windows or Unix/Linux binary libraries. Among the emerging open-source options, the level of documented verification, certification and testing tends to be somewhat limited. The limited availability of optimization software suitable for embedded MPC has led to several recent research activities that target open source and library-free portable numerical optimization code using a convenient programming language and style such as subsets of C and C++ (avoiding e.g., dynamic memory allocation, and pointers). This software enables embedded MPC implementation, but as formal verification and documentation of extensive testing may not always be available, the question of dependability still remains somewhat open. The developments can be categorized according to optimization algorithm type as the following.

1) **Active set convex quadratic programming**. The QPOASES algorithm takes advantage of the fact that the time-varying parameters of the MPC problem (such as states and setpoints) typically leads to a quadratic program, where the time-varying parameters only appear linearly in the constraints, and that they typically make small changes from one sample to the next [28]. QPOASES searches for the optimal active set by moving through the parameter space, and inherently includes a warm-start procedure. The original implementation was made in C++, but C-versions now exist.
2) **Interior point convex/quadratic programming**. Various interior point methods have been implemented as automatically generated customized C code solvers, i.e., CVXGEN [29] and FORCES [30] and [31]. An effective variant of an interior point method that uses Riccati iterations to compute the gradient is presented in [32].
3) **Fast gradient convex/quadratic programming**. Fast gradient methods are first order optimization methods that perform weighted/filtered gradient descent steps and constraint projections, usually taking advantage of the very simple projection needed to solve the dual problem (i.e., positive Lagrange multipliers) for constraint fulfillment [4], [33]–[35]. In addition to very simple code that leads to a small footprint and may allow formal verification of the code, one can bound the number iterations needed to achieve a given accuracy.
4) **Nonlinear programming**. The ACADO software tool contains methods for discretization of continuous-time nonlinear optimal control problems and numerical solution of the resulting nonlinear program [36], [37]. The software exploits symbolic computation of gradients based on a C++ specification of the model, constraints and objective. In addition to the basic C++ version, efficient C code can be automatically generated for embedded system implementations.

Embedded MPC applications tend to require fast update intervals within a highly resource-limited computer environment. Although the size of embedded MPC control problems tends to be significantly smaller than in conventional MPC, and warm start procedures can be used, one cannot expect that the optimization solver will be allowed to execute until strict tolerances on the optimality conditions or stopping criteria are met in every case. Hence, in order to fulfill the strict requirements for real-time processing, only a finite (usually small) number of iterations is allowed. Theoretical studies have provided theories, stability criteria and suboptimality (performance) bounds that cover such limitations, e.g., [38]–[43]. These theories are primarily of conceptual value as their direct applicability is limited. First, assumptions may not be theoretically guaranteed or verified *a priori* in a given application, or the potentially adverse impact of uncertainty on models and measurements. Second, the benefits of a long prediction horizon would require more computational resources.

Embedded processing units are often highly resource limited, and it has been demonstrated that efficiency of the implementation of embedded MPC not only depends on the chosen numerical algorithm and the HW platform but also on the implementation in software. For example, in [44], it is shown that significant improvements in computational speed can be achieved by considering the system memory architecture and exploit in the numerical linear algebra computations the device's

registers, fast cache, data addressing modes, etc. It is also experienced that generating branch-free code with unrolled loops (e.g., [29]) may lead to efficient code execution at the cost of a larger memory footprint.

Extensive effort has been invested into the research for algorithms that can precompute and store the solution to the MPC problem as a function of its input parameters (such as current state, objective, and constraints). While some useful algorithms and software to compute so-called explicit MPC have been developed, e.g., [45]–[47], their practical use is limited to fairly small-scale applications, where there is no need for an online update of setpoints, constraints, and models. The embedded system implementation typically consists of some data structure that stores the precomputed solution and some algorithm that can efficiently search through this data structure for the current optimal MPC solution, [48]–[50]. This approach requires very limited computational resources and no need for floating-point computations.

From an implementation point of view, a clear distinction goes between linear and nonlinear MPC, or more precisely, between convex and nonconvex optimization. While linear MPC requires numerical linear algebraic computations (which also can be very challenging and lead to failures of the MPC if not implemented in a suitable manner), nonlinear MPC requires, in addition, the accurate computation of gradients and may come with additional challenges such as lack of convexity (manifested through the existence of multiple local minimums, indefinite Hessians, etc.), possibly lack of smoothness of the cost and constraints, and numerous numerical issues. While linear MPC provides a reasonably "closed" class of problems for which general solutions can be made, there seems to be little hope to be able to make any general solutions for nonlinear MPC except for certain classes of systems and problems. A state-of-the-art software tool for embedded nonlinear MPC is ACADO [36]. Real-time performance is achieved by a so-called real-time iteration approach that precomputes data for the optimization algorithm based on the previous step while waiting to receive new state measurements. This reduces the latency from state measurements received until the control input is computed.

III. Dependable MPC

This section discusses dependability in the context of MPC, using standard taxonomy and concepts [15]. The first section deals with requirements and assessment of dependability of an embedded MPC. The second section analyzes potential faults, errors, failures, and their consequences in embedded MPC, whereas the third section discusses some general means to improve dependability in embedded MPC.

A. Requirements and Assessment of Dependable Embedded MPC

The general attributes of system dependability are reliability, availability, safety, integrity, and maintainability, [15]. In the context of embedded MPC, the aspects of *safety* and *reliability* are key and therefore given special attention here. The other attributes are not discussed any further, as they can be assumed to be managed through industry standard embedded system design and development procedures without much special consideration for MPC.

Safety requirements of a system need careful analysis that may go far beyond the functionality of the MPC itself. The implications of safety requirements on an MPC design and implementation will depend strongly on the control system architecture, as illustrated for some possible cases in Fig. 2, and the characteristics of the system under control and its environment. Reliability refers to the continued correctness of functionality. Some of the concepts involved are much the same as aforementioned in the context of safety. However, the requirements can in many cases be different as in some systems safety and performance are treated separately during design and development, whereas, in other systems, they are all part of an integrated design and development process. Some common safety and reliability requirements specific to MPC can typically be characterized in terms of

1) A description of acceptable MPC faults and errors that can be handled by its inherent fault-tolerance and robustness [51], or by the basic control system and safety system. For example, MPC may be designed as a fail-controlled system that can fail in specific failure modes, [52]. If independent basic control or safety systems are not available, one will expect that less faults and errors within the MPC can be acceptable. Further discussion on typical issues is given in Section III-B.
2) Requirements for internal fault detection, diagnosis, reconfiguration, and accommodation within the MPC, and accompanying inputs, diagnostic outputs, and alarms. Further discussion of relevant methods and tools are given in Section III-C.
3) For a characterization of acceptable mean time between faults, fault probability, or similar reliability requirements that can be used for design and analysis such as verification, validation, testing, and certification procedures, see also Section III-C.

The MPC objective and performance is generally specified in terms of a cost function and a set of constraints. Typically, there is an explicit or implicit priority hierarchy underlying this specification as some of the objectives and constraints are more important than others. The implementation of this hierarchy of objectives and constraints leads to a selection of weights and infeasibility handling mechanisms that usually solves a sequence of optimization problems in order to not violate high priority constraints unless strictly needed. Commonly, the most highly prioritized constraints are characterized as safety constraints, whereas others are not considered to be safety critical. Hence, in a given situation, the MPC may predict that certain safety constraints are likely to be violated at some point in the near future. While this is useful information for alarms, plant reconfiguration, shutdown and emergency actions, one need to have in mind that the predictions are subject to uncertainty due to modeling and measurement errors. Although robust versions of MPC might build in margins toward uncertainty in the optimization problem, this generally comes at a price

of conservativeness and higher computational complexity. In many applications, requirements to performance, availability and reliability means that unnecessary shutdown or alarms should be avoided. In this context, safety, reliability, and availability may be conflicting objectives since sometimes one may be improved only at the cost of reducing one of the others.

B. Potential Faults, Errors, and Failures

We start by describing some typical faults and errors, and then continue with some discussion on how these faults can lead to MPC failure.

A known MPC design issue is whether to allow constraints that might prevent a feasible solution from existing. Infeasibility of the optimization problem typically leads to a failure of the MPC, as the software will not be able to return a valid or useful solution. In some applications, in particular with short prediction horizons and open-loop unstable processes, the use of terminal constraints may be used to ensure stability properties of the closed loop control system, [2]. However, infeasibility of the terminal constraint along the predicted state trajectory does not necessarily mean that instability will result as the terminal constraint formulation may be conservative, the model is uncertain, or disturbances may be more favorable than anticipated (one may be lucky!). Likewise, the violation of some other state constraints along the predicted state trajectory may also be acceptable (as a last resort) or may actually not happen due to prediction errors.

One can therefore argue that, in any case, it makes sense to formulate the optimization problem with slack variables on all constraints (except perhaps input constraints that are physical limitations) or use an explicit constraint priority hierarchy to support relaxation of the least prioritized constraints when necessary. Then one can guarantee that a feasible solution will always exist. Both approaches increase the complexity of the MPC optimization. Slack variables result in increased dimension and range of numerical values of weights. With a constraint priority hierarchy, there are additional optimization problems that need to be solved in order to determine which constraints must be relaxed. One should have in mind that the use of hard constraints only on input variables means that it will be fairly straightforward to guarantee that a feasible solution is found (also initially before any iterations are made), even for highly nonconvex problems. The reason is that the input variables are usually directly related to the variables being optimized by an MPC. Nonzero slack variables or relaxed constraints are indicators or warnings about likely future constraint violations, possible instability, or other adverse conditions. Expected sources of constraint violations are large disturbances or challenging references for the controller. There are physical limitations in the process plant or disturbances that makes constraint violation unavoidable in certain conditions.

Assuming that the MPC problem is formulated such that a feasible solution always exists, one still needs to worry about the performance and optimality of the "solution" returned by the numerical optimization software. Failure to meet optimality conditions may typically be caused by algorithmic faults, software implementation errors, resource limitations (no time to run the necessary number of iterations), lack of convexity (in nonlinear MPC), numerical inaccuracies, e.g., in the computation of gradients, degeneracies in the problem that are not well accounted for in the implementation, or other design or implementation errors. The accumulation of numerical round-off errors in iterative numerical algorithms is usually counteracted with robust numerical methods using double-precision floating-point numeric data representation. In embedded systems based on very simple processors, even robust numerical algorithms may have problems if fixed-point (integer) numerical data representations are used in order to reduce the computational cost compared with computationally expensive single- or double-precision floating-point emulation using software. Particular number and processing systems may be designed in order to optimize the numerical performance, e.g., [24].

For nonlinear MPC, the typical numerical solvers are not able to distinguish a local optimum from a global optimum. In some problems with nonconvex objective or constraints, local optimality may not be sufficient to achieve the required performance.

Performance degradation must be expected under certain fault conditions, as those discussed previously here. How can this be detected automatically? What are the consequences and severities of failures? One need to distinguish between minor failures (that are typically not safety critical) and catastrophic failures having cost of consequences that may be much larger than the benefits of the MPC.

In embedded MPC systems that are expected to operate without interruption for extended periods of time, the issue of "software aging" is relevant. It refers to progressively accrued conditions resulting in performance degradation or failure. Examples are memory bloating and leaking, unavailable resources that were not properly released when they should, data corruption/overwriting, storage space fragmentation, and accumulated numerical round-off errors. Common features of several recent optimization algorithm implementations intended for embedded systems are avoiding dynamic memory allocation and careful protection of the memory space.

In addition to the aforementioned faults related to design, implementation and operation of the system, one should also be concerned about specification faults, commissioning faults, incomplete analysis and design, misinterpretations, unwarranted assumptions, documentation errors, inconsistencies, etc. This is part of general engineering and control design and not specific to embedded MPC; thus, we do not treat it further here.

C. System Design and Development

Best practice for development of dependable embedded computing includes the following approaches, [15], which are found to be effective in the context of MPC.

1) **Fault prevention**. In the system design and development, one can prevent faults by robust algorithm design that uses recursive numerical methods that will not accumulate the effects of round-off errors and avoid potentially illegal or sensitive numerical operations. A key success factor in iterative numerical optimization is robust initialization utilizing quality-assured measurements, results

of previous successful iterations or solutions, and application knowledge to prevent faulty initialization. The choice of software (SW) and hardware (HW) architecture should be based on ultrareliable industrial HW and SW environments that provide reliable hardware under a sufficient range of environmental conditions. The embedded system SW environment should avoid dynamic memory allocation, use strict typing, and memory protection.

2) **Fault tolerance**. In the system design and implementation, a common and effective approach involves HW and SW redundancy. Several industrial computer environments have firmware support for automatic switchover to a hot redundant processor in case of fault or failure. SW redundancy may include multiple optimizations and fallback solutions in case of SW failure.

   For example, in [53] the fallback solution is a relatively simple Hildreth's numerical optimization algorithm that is executed in parallel with a more high-performance solver. In general, diversity (independent solutions in design and/or implementation) may lead to increased fault tolerance as certain common mode failures can be avoided. The use of post-optimal analysis, exception handling, and multithreaded programming may prevent faults to lead to failures as the fault-free part of the system may continue without being impacted. In MPC, monitoring of constraints and their violation/margin is commonly used in industrial implementation and model errors might lead to unacceptable gaps that should be corrected for by updating models through online estimation. Fault-hardening of software implementation is studied in [52].

3) **Fault detection and identification**. It is a great advantage if internal MPC function faults be detected, identified and signaled as diagnostic information to the rest of the system (including operator, engineers, basic control, and safety system). Many numerical optimization solvers provide return codes and warnings that can be used for this purpose, and additional specific mechanisms can also be implemented [52].

4) **Fault removal**. Various methods for software and system testing and verification are effective, such as simulator-based testing using dedicated fault-injection tools, [52], [54]. This also includes, in particular, the use of hardware-in-the-loop (HIL) simulation-based testing. Hot (periodic) reset may be effective to avoid the effect of "software aging".

5) **Fault prediction**. MPC includes an inherent dynamic mathematical model of the system, which means that simulation is an effective tool in order to not only verify and test the system, but also to predict which faults or failure modes can be expected, and what are their consequences for the system, [52]. While HIL-testing and fault-injection tools [54] are useful also here, its effectiveness is limited by the fact that the system operates in real time, which means that HIL-testing would be slow compared with software simulation on a faster computer system. The faults are likely to be very rare events; thus, specially tailored scenarios would be needed to complement Monte Carlo simulations and other exhaustive techniques.

MPC requires a very complex software and development processes, and error-free software may be an impossible goal, although numerical methods based on fast gradient methods (as discussed in Section II-C) may enable formal software verification as it provides small footprint, simple code, theoretical convergence and error bounds. As in many other cases, [55], one could also focus on ways to build MPC software and systems that are robust and safe in the presence of typical software errors. A control system will need to face uncertainty in mathematical prediction models and physical equipment and materials; thus, "perfect software" is in any case not a sufficient technical requirement for safety and reliability.

As a partial conclusion, we find that with a complex numerical algorithm as the basis for the embedded MPC it is in general difficult to assess, based on the optimization software output if the computed control action is reliable (correct) and safe, or not, and how well it will perform. The reasons for this difficult assessment is prediction uncertainty (due to unknown disturbances and modeling and measurement errors), difficulty of interpreting slack variables and other auxiliary outputs and diagnostic information from the optimization software, possible existence of local minimums, as well as potential software errors in the optimization software. This leads to the critical question, *how to detect faults in the MPC and assess their potential for future failure of the control to meet the control objectives and constraints?* Some suggestions and new results in this direction are provided in Section IV.

IV. Proposed Architecture for Dependable Embedded MPC

As discussed in Section III-C, there are several means to ensure the dependability of embedded MPC. Different techniques are likely to be favorable in different applications and based on the dependability requirements. The proposed MPC system architecture is therefore not intended to be a recommended approach for every embedded MPC, but still believed to be sufficiently flexible to be useful to a range of applications and implementations.

A. Functional Redundancy

The basic idea is to build an architecture for dependable embedded MPC on the principle of functional redundancy. This is illustrated in Fig. 3, where multiple control algorithms or implementations propose control alternatives that are evaluated and compared by a separate algorithm called performance monitor and control selection, which eventually selects the single control alternative that is expected to achieve the best control performance (in terms of the MPC cost function and constraints). The following observations and comments can be made.

1) Assuming that the Performance Monitor and Control Selection algorithm is able to make a correct decision (see Section IV-B for further descriptions and discussion on this important nontrivial issue), it is sufficient that one of the control alternatives provides an acceptable solution. This is in particular the role of the Backup Controller illustrated in Fig. 3, and is similar to the Simplex architecture [56], [57], which has both a safety controller and a baseline controller that together serves such a purpose.

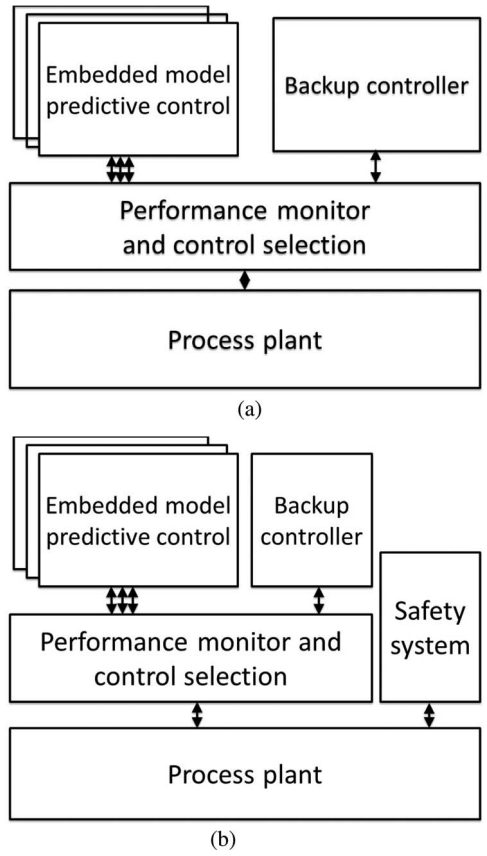

Fig. 3. Two versions of a proposed resilient embedded MPC system architecture. (a) Resilient embedded MPC system architecture. (b) Resilient embedded MPC system architecture with independent safety system.

2) The performance monitor and control selection process should not introduce any significant latency or delay in the execution of the control action. This means that the calculations should be fast and executed immediately after the MPC calculations such that the control selection can be made before a new cycle is scheduled.
3) One can think of the performance monitor as making post-optimal analysis. It can base its analysis on solver return status that may typically indicate if local optimality conditions are fulfilled or not, and if some error conditions have occurred during optimization. In addition, the performance monitor can make its own independent assessment of the fulfillment of the MPC performance and constraint satisfaction. As described in Section IV-B, the use of an independent faster than real-time simulator could be a useful tool for this. This may be particularly important for nonlinear/nonconvex MPC, where fulfillment of local optimality conditions may not be sufficient to achieve the desired performance, or the solver return status cannot be fully trusted.
4) The redundancy can be realized in different ways, e.g., use time-shifted optimal solution from the previous solution (as theoretically analyzed in e.g., [43]), use multiple optimization solvers, [53], one solver with different settings and options, multiple initializations or warm start procedures, use multiple MPC formulations (possibly with relaxed constraints, simplified models, perturbed parameters in order to avoid degeneracies, etc.), or simpler "backup controller" solutions such as PID or linear quadratic regulator. As a minimum, two sufficiently independent control alternatives are needed, although more diversity though multiple MPC instances or algorithms may be considered favorable at the cost of additional computational complexity.
5) Some level of performance degradation (compared with an optimal solution) should be considered acceptable within the proposed system architecture. Upon nonacceptable performance degradation (e.g., violation of certain constraints) of all available control alternatives, shutdown, or fail-to-safe functions might be executed.
6) The proposed redundancy may be realized only in software or in a combination of hardware and software. This choice may depend on the system requirements, reliability of the control hardware, and the ability of the hardware to catch software faults and exceptions, including real-time requirements. As aforementioned, the correctness and reliability of the performance monitor and control selection is essential for the dependability of the redundant embedded MPC architecture.
7) The redundant control alternatives can be evaluated using parallel processing and can exploit multicore or multiprocessor architectures [53], [58], as well as cloud computing techniques. The performance monitoring can also be partly executed through parallel independent evaluations, but the final comparison and control selection may not be made until all control alternatives have been evaluated. Hence, with parallel processing there will still be a small coordination overhead.
8) Switching between controllers is known to degrade performance (chattering) and stability in some cases. However, since the switching criterion should be performance-based (simulation over a sufficiently large horizon into the future, always using the same performance criterion), there seem to be no strong instability mechanisms inherent in the proposed approach. However, further investigation and theoretical research on this issue would be needed in order to firmly establish some conclusions.
9) Risk-based verification scope management as proposed in [59] is applicable to analyze how verification resources are most efficiently utilized in this approach. It requires that the losses due to failure, as well as effectiveness of verification and testing can be estimated. It is quite straightforward and intuitive to see that when the Backup Controller is simple and provides performance not too suboptimal compared with the MPC, it may be favorable to invest verification resources on the backup controller and the performance monitor and accept the reasonable high fault rate of the MPC. On the other hand, when the backup controller leads to significant losses or risks compared with the MPC, it might be favorable with a shift of verification effort to in the MPC module.

B. Simulator-Based Performance Monitor

The task of the performance monitor is to predict which of the alternative control trajectories will provide the best per

formance. While conventional control performance monitoring techniques considers the statistics of recent historic data, or the current state of the system [56], [57], our need to avoid the consequences of faults typically requires a prediction of future performance. This is a more difficult task, which we will base on a faster than real-time simulation of the alternative control trajectories, although an analysis of recent historic data may provide a useful supplement.

We note that the selection of simulation model and numerical simulation accuracy are crucial, as MPC control trajectories are optimized by solving an open-loop optimal control problem. Predicting the performance of the open-loop optimal control by simulating its response with a different model or disturbances would potentially strongly overestimate the deviation since the effect of feedback achieved by reoptimizing the control trajectory at the next sample is not accounted for. Unfortunately, performance prediction under closed loop conditions with the actual MPC feedback control would typically not be possible to implement in real time in a resources-limited environment because a sequence of MPC problems would need to be solved as part of this simulation. Hence, in order to implement the performance monitor with a faster than real-time simulation and avoid steady-state differences, one should take care to ensure that future disturbances are set up in the same way in the simulator as in the MPC predictions. This, on the other hand, might lead to deviations being underestimated since the effects of model uncertainty and disturbances on the MPC are optimistic. When relevant, there should be implemented mechanisms to monitor local versus global optima, as well as to monitor fulfillment of constraints in order to account for modeling/prediction errors.

A pseudocode summary of a simulator-based performance monitor and control selection is given in Algorithm 1.

while *true* **do**
1. Get the current state;
2. Get disturbance estimates;
3. Predict future disturbance;
4. Execute the MPC calculations;
5. Simulate the MPC control trajectories in open loop on the prediction horizon;
6. Simulate the backup controller in closed loop on the prediction horizon;
7. Evaluate all results with priorities: Constraint violations, control performance (cost function), and other diagnostic information (e.g. operating system, hardware monitoring, fault detection and diagnostic modules, optimizer status output, etc.) ;
8. Rank the acceptable solution;
9. Select the best solution;
10. Scheduled the best solution for use;
11. Issue alarms and other diagnosis;
12. Use best trajectory for warm start;

end

C. Simulation Example

In order to illustrate the proposed Simulation-based Performance Monitoring and Control Selection algorithm and its benefits when used in the architecture in Fig. 3(a), we consider a simulation example. The MPC has the objective of controlling the angle of attack and pitch angle using elevator and flaperon inputs of an unstable aircraft. Thus, the system has two inputs (u_1 and u_2) and two outputs (y_1 and y_2), with constraints on all variables. The control problem formulation and fourth-order linearized state space model is [60], and the MPC cost function defined at time t_0 is

$$J(t_0) = \sum_{i=0}^{N} w_1 \left(y_1(t_{i+1}) - r_1\right)^2 + w_2(y_2(t_{i+1}) - r_2)^2 + p_1 \Delta u_1(t_i)^2 + p_2 \Delta u_2(t_i)^2 \quad (1)$$

where $N = 10$ is the prediction horizon (corresponding to 0.5 s), r_1 and r_2 are time-varying reference values, $\Delta u(t)$ denotes change since previous sample, and cost function weights $p_1 = p_2 = 0.5$, $w_1 = 8$ and $w_2 = 20$. In addition the MPC takes into account constrains on the horizon in the following form, for $i = 0, 1, \ldots N$:

$$-25^\circ \leq u_1(t_i), u_2(t_i) \leq 25^\circ, \quad -5^\circ \leq y_1(t_i), y_2(t_i) \leq 5^\circ.$$

As the backup controller, the linear controller is used from [61] that considered the same example. In order to implement a simulation-based performance monitor, we simulate the nominal system with the optimal future control trajectory provided by the MPC and record its predicted performance through the cost function (1), denoted $J_{\mathrm{MPC}}(t_i)$. Likewise, we simulate the nominal system in closed loop with the backup controller and record its predicted performance on the prediction horizon through the cost function (1), denoted $J_{\mathrm{backup}}(t_i)$. The MPC controller is selected for use if it fulfills the following criterion:

$$J_{\mathrm{MPC}}(t_i) < (1+\gamma) J_{\mathrm{backup}}(t_i) \quad \text{OR} \quad J_{\mathrm{MPC}}(t_i) < J_{\min}.$$

If not, the backup controller is selected. The tuning parameters are $\gamma > 0$ and $J_{\min} > 0$, which in our simulations are chosen as $\gamma = 0.1$ and $J_{\min} = 20$ as a robust tradeoff to prevent undesired switching due to effects of noise and disturbances.

The first failure mode we simulate is that the MPC output freezes and stays constant for the period $2.5 \leq t \leq 4$. Typical faults leading to this failure mode are faults in the software code, task, or computer executing the MPC algorithm, but it could also be due to I/O, interface or communication faults or latencies in the embedded system. While, in some cases, there would be other indicators or status information that would give a clue about the fault, we consider in this example only the aforementioned simulation-based performance assessment criterion. The simulation results are shown in Fig. 4, where we have added pseudorandom white noise disturbances representing turbulence and measurement noise. We observe that the frozen MPC output is partially accepted during the period $2.5 \leq t \leq 3.0$ since the flight conditions are stable and constant control input is most of the time sufficient for this short period of time until the set-point changes at $t = 3.0$. At this point, the inferior performance of the MPC is quickly detected and the control selector switches to the backup controller. Upon recovery of the output from the MPC at $t = 4.0$, the superior performance of the MPC is detected and the control selector switches back from the backup controller to the MPC. Additional simulations reveal that due to the open-loop unstable nature of the aircraft, complete loss of stability of the aircraft would have

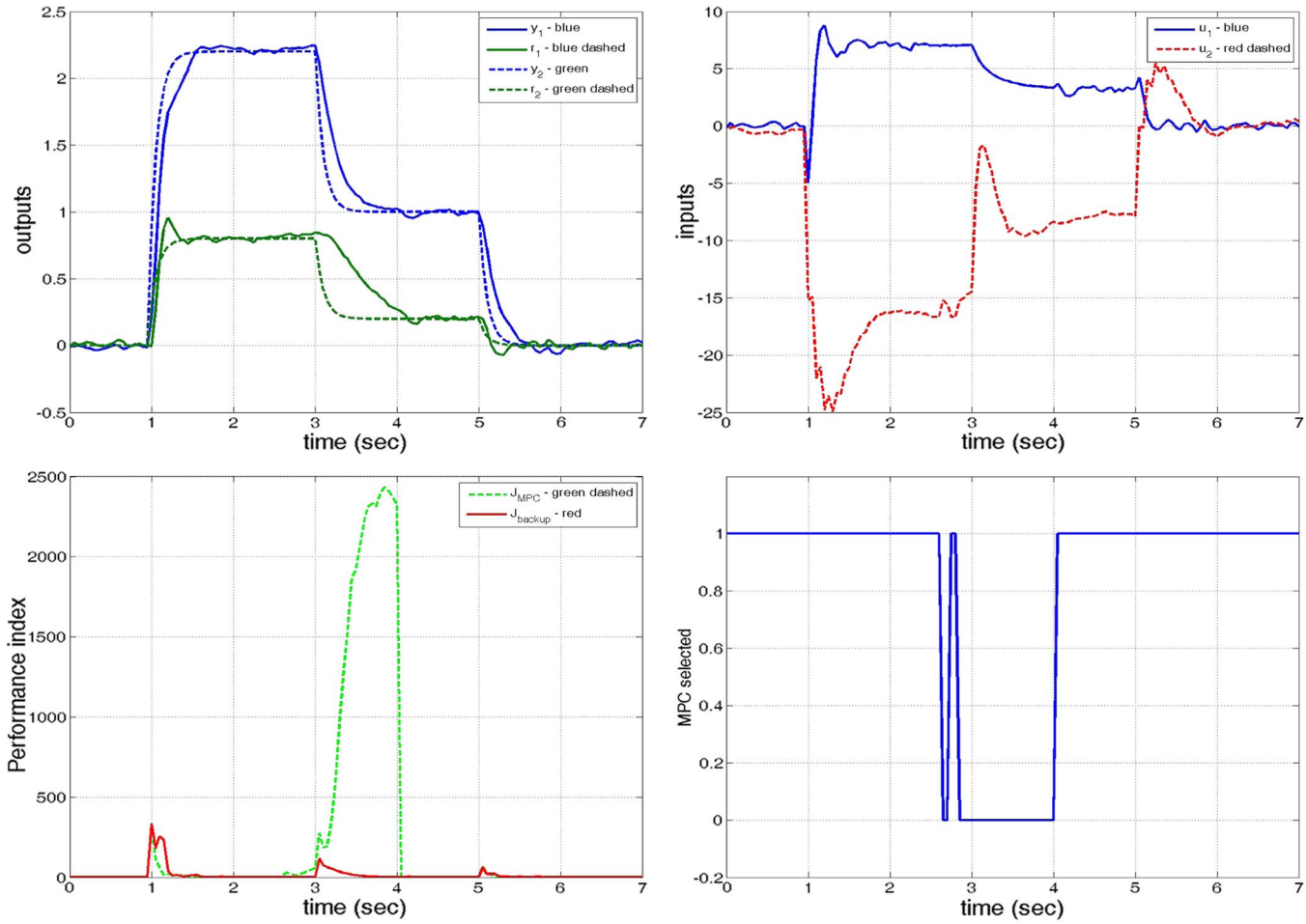

Fig. 4. Simulations with MPC freeze failure mode occurring during $2.5 \leq t \leq 4$ s.

occurred after less than 1 s without switching to the backup controller in this scenario.

The second failure mode we simulate is that the MPC output is inaccurate for the period $2.5 \leq t \leq 4$. More specifically, we simulate this by adding normally distributed pseudorandom numbers with standard deviation 2° to the input trajectories computed by the MPC. Typical faults leading to this failure modes are similar to the aforementioned case, but of intermittent and less severe character. The simulation results are shown in Fig. 5. We observe the inferior performance of the MPC is quickly detected and the control selector switches to the backup controller immediately at $t = 2.5$. Upon recovery of the output from the MPC at $t = 4.0$, the superior performance of the MPC is detected and the control selector switches back to the MPC. Additional simulations show that in this case, the MPC performance degradation is not severe enough to cause instability of the aircraft if not switching to the backup controller, but leads to unsteady behavior and generally poor performance.

V. Conclusion

The technology of embedded MPC has emerged quickly over the last decade, with open source software supporting the implementation of online numerical optimization and optimal data structures precomputed by offline parametric programming on highly resource-limited embedded hardware and software architectures. In particular, the use of fast gradient methods for numerical optimization in linear MPC show good promise for embedded systems due to their small footprint, computational efficiency, theoretical convergence guarantees, and simple code that may admit formal verification.

The need for verification of the embedded MPC may, in some applications, be relaxed as safety may be accounted for by building a resilient architecture around the MPC with redundant functionality and automatic control performance monitoring. Still, the effects of model uncertainty and unknown disturbances on the control performance are fundamental limitations.

Some important future research challenges are related to tighter performance bounds on numerical algorithms, allowing the effect of limiting the number of iterations to be predicted in an accurate nonconservative manner. For nonlinear MPC, there are many additional challenges due to possible lack of convexity and regularity of the mathematical programming problem. Improved tool-chains with methods for automatic code design, system integration, and verification of the resulting implementation would enable more widespread industrial use of embedded MPC.

Acknowledgment

The author would like to thank S. Ruud and A. Karlsen at DNV GL and G. O. Eikrem and A. Pavlov at Statoil for the contributions through many interesting discussions.

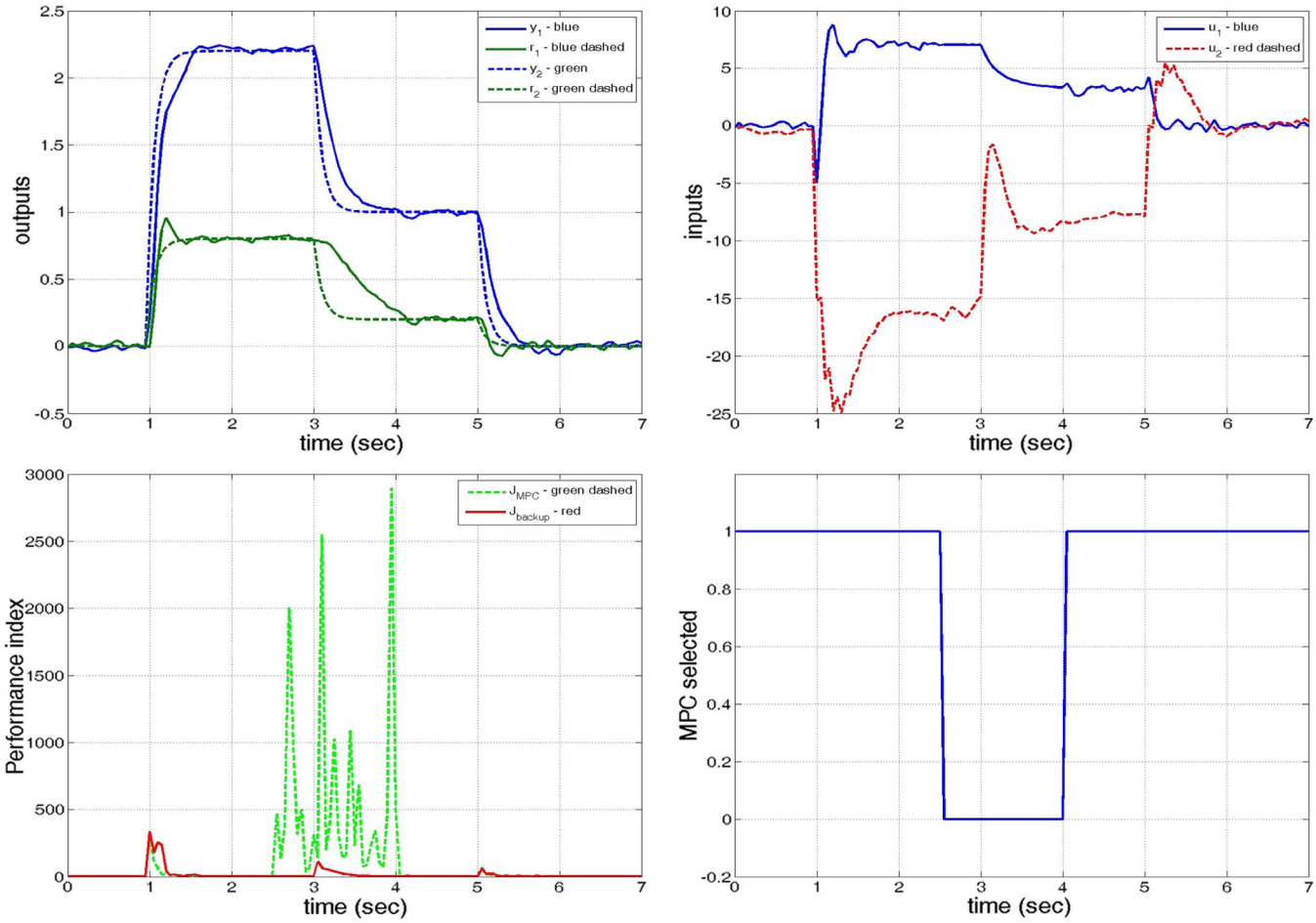

Fig. 5. Simulations with MPC inaccurate output occurring during $2.5 \leq t \leq 4$ s.

REFERENCES

[1] S. J. Qin and T. A. Badgwell, "A survey of industrial model predictive control," *Control Eng. Pract.*, vol. 11, no. 7, pp. 733–764, Jul. 2003.

[2] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, Jun. 2000.

[3] J. B. Rawlings and R. Amrit, "Optimizing process economic performance using model predictive control," in *Nonlinear Model Predictive Control—Towards New Challenging Applications*, vol. 384, *Lecture Notes in Control and Information Sciences*, L. Magni, D. M. Raimondo, and F. Allgöwer, Eds. Berlin, Germany: Springer-Verlag, 2009.

[4] D. K. M. Kufoalor *et al.*, "Embedded model predictive control on a PLC using a primal-dual first-order method for a subsea separation process," presented at the Proc. Mediterranean Conf. Control Automation, Palermo, Italy, 2014, Paper TuCT2.1.

[5] B. J. T. Binder, D. K. M. Kufoalor, A. Pavlov, and T. A. Johansen, "Embedded model predictive control for an electric submersible pump on a programmable logic controller," presented at the Proc. IEEE Multi-conference Systems Control, Nice, France, 2014, Paper WeC01.2.

[6] O. Breyholtz, G. Nygaard, and M. Nikolaou, "Managed-pressure drilling: Using model predictive control to improve pressure control during dual-gradient drilling," *SPE Drilling Completion*, vol. 26, no. 2, pp. 182–197, 2011.

[7] L. van den Broeck, M. Diehl, and J. Swevers, "A model predictive control approach for time optimal point-to-point motion control," *Mechatronics*, vol. 21, no. 7, pp. 1203–1212, Oct. 2011.

[8] B. Acikmese *et al.*, "Flight testing of trajectories computed by g-fold: Fuel optimal large divert guidance algorithm for planetary landing," presented at the Proc. 23rd AAS/AIAA Spaceflight Mechanics Meeting, Kauai, HI, USA, 2013, Paper AAS 13-386.

[9] T. I. Bø and T. A. Johansen, "Dynamic safety constraints by scenario based economic model predictive control," in *Proc. IFAC World Congr.*, Cape Town, South Africa, 2014, pp. 9412–9418.

[10] C. Chapman, M. Musolesi, W. Emmerich, and C. Mascolo, "Predictive resource scheduling in computational grids," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2007, pp. 1–10.

[11] D. E. Quevedo, R. P. Aguilera, and T. Geyer, "Predictive control in power electronics and drives: Basic concepts, theory, methods," in *Advanced and Intelligent Control in Power Electronics and Drives*, vol. 531, *Studies in Computational Intelligence*, T. Orowska-Kowalska, F. Blaabjerg, and J. Rodriguez, Eds. Dordrecht, Switzerland: Springer-Verlag, 2014, pp. 181–226.

[12] C. R. Gutvik, T. A. Johansen, and A. O. Brubakk, "Optimal decompression of divers—Procedures for constraining predicted bubble growth," *IEEE Control Syst. Mag.*, vol. 31, no. 1, pp. 19–28, Jan. 2011.

[13] P. Falcone, F. Borrelli, E. H. Tseng, and D. Hrovat, "On low complexity predictive approaches to control of autonomous vehicles," in *Automotive Model Predictive Control*, *Lecture Notes in Control and Information Sciences*. London, U.K.: Springer-Verlag, 2010, pp. 195–210.

[14] D. K. Kufoalor and T. A. Johansen, "Reconfigurable fault tolerant flight control based on nonlinear model predictive control," in *Proc. Amer. Control Conf.*, Washington, DC, USA, 2013, pp. 5128–5133.

[15] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33, Jan.–Mar. 2004.

[16] R. C. Shekhar and J. M. Maciejowski, "Robust predictive control with feasible contingencies for fault tolerance," in *Proc. IFAC World Congress*, Milano, Italy, 2011, pp. 4666–4671.

[17] E. Camacho, T. Alamo, and D. de la Pena, "Fault-tolerant model predictive control," in *Proc. IEEE Conf. ETFA*, 2010, pp. 1–8.

[18] L. Lao, M. Ellis, and P. D. Christofides, "Proactive fault-tolerant model predictive control," *AIChE J.*, vol. 59, no. 8, pp. 2810–2820, Aug. 2013.

[19] N. G. Leveson, *Engineering a Safer World: Systems Thinking Applied to Safety*., 1st ed. Cambridge, MA, USA: MIT Press, 2011.

[20] B. Huyck *et al.*, "Towards online model predictive control on a programmable logic controller: Practical considerations," *Math. Probl. Eng.*, vol. 2012, pp. 912 603:1–912 603:20, 2012.

[21] P. Zometa, M. Kogel, T. Faulwasser, and R. Findeisen, "Implementation aspects of model predictive control for embedded systems," in *Proc. Amer. Control Conf.*, Montreal, QC, Canada, 2012, pp. 263–275.
[22] M. He and K.-V. Ling, "Model predictive control on a chip," in *Proc. Int. Conf. Control Autom.*, Budapest, Hungary, 2005, pp. 528–532.
[23] L. Bleris, J. Garcia, M. V. Kothare, and M. G. Arnold, "Towards embedded model predictive control for system-on-chip applications," *J. Process Control*, vol. 16, no. 3, pp. 255–264, Mar. 2006.
[24] P. Vouzis, L. Bleris, M. Arnold, and M. Kothare, "A system-on-a-chip implementation of embedded real-time model predictive control," *IEEE Trans. Control Syst. Technol.*, vol. 17, no. 5, pp. 1006–1017, Sep. 2009.
[25] K. Ling, B. Wu, and J. Maciejowski, "Embedded Model Predictive Control (MPC) using a FPGA," in *Proc. IFAC World Congr.*, Seoul, Korea, 2008, pp. 15 250–15 255.
[26] J. Jerez, K.-V. Ling, G. Constantinides, and E. Kerrigan, "Model predictive control for deeply pipelined field-programmable gate array implementation: Algorithms and circuitry," *IET Control Theory Appl.*, vol. 6, no. 8, pp. 1029–1041, May 2012.
[27] A. G. Wills, G. Knagge, and B. Ninness, "Fast linear model predictive control via custom integrated circuit architecture," *IEEE Trans. Control Syst. Technol.*, vol. 20, no. 1, pp. 59–71, Jan. 2012.
[28] H. J. Ferreau, H. G. Bock, and M. Diehl, "An online active set strategy to overcome the limitations of explicit MPC," *Int. J. Robust Nonlin. Control*, vol. 18, no. 8, pp. 816–830, May 2008.
[29] J. Mattingley and S. Boyd, "CVXGEN: A code generator for embedded convex optimization," *Optim. Eng.*, vol. 13, no. 1, pp. 1–27, Mar. 2012.
[30] A. Domahidi, A. Zgraggen, M. Zeilinger, M. Morari, and C. Jones, "Efficient interior point methods for multistage problems arising in receding horizon control," in *Proc. IEEE Conf. Decision Control*, Maui, HI, USA, 2012, pp. 668–674.
[31] A. Shahzad and P. J. Goulart, "A new hot-start interior-point method for model predictive control," in *Proc. 18th IFAC World Congr.*, Milano, Italy, 2011, pp. 74–79.
[32] G. Frison, H. H. B. Sørensen, B. Damman, and J. B. Jørgensen, "High-performance small-scale solvers for linear model predictive control," in *Proc. 13th ECC*, Strasbourg, France, 2014, pp. 128–133.
[33] C. N. Jones *et al.*, "Fast predictive control: Real-time computation and certification," in *Proc. IFAC Conf. Nonlin. Model Predictive Control*, Noordwijkerhout, The Netherlands, 2012, pp. 94–98.
[34] P. Patrinos and A. Bemporad, "An accelerated dual gradient-projection algorithm for embedded linear model predictive control," *IEEE Trans. Autom. Control*, vol. 59, no. 1, pp. 18–33, Jan. 2014.
[35] I. Necoara and V. Nedelcu, "On linear convergence of a distributed dual gradient algorithm for linearly constrained separable convex problems," Univ. Politehnica Bucharest, Bucharest, Romania, Tech. Rep., Oct. 2013.
[36] B. Houska, H. J. Ferreau, and M. Diehl, "An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range," *Automatica*, vol. 47, no. 10, pp. 2279–2285, Oct. 2011.
[37] B. Houska, H. J. Ferreau, and M. Diehl, "ACADO toolkit—An open-source framework for automatic control and dynamic optimization," *Optim. Control Appl. Methods*, vol. 32, no. 3, pp. 298–312, May/Jun. 2011.
[38] P. O. M. Scokaert, D. Q. Mayne, and J. B. Rawlings, "Suboptimal model predictive control (feasibility implies stability)," *IEEE Trans. Autom. Control*, vol. 44, no. 3, pp. 648–654, Mar. 1999.
[39] G. Valencia-Palomo and J. A. Rossiter, "Efficient suboptimal parametric solutions to predictive control for plc applications," *Control Eng. Pract.*, vol. 19, no. 7, pp. 732–743, Jul. 2011.
[40] S. Richter, C. N. Jones, and M. Morari, "Computational complexity certification for real-time MPC with input constraints based on the fast gradient method," *IEEE Trans. Autom. Control*, vol. 57, no. 6, pp. 1391–1403, Jun. 2012.
[41] S. Richter, C. N. Jones, and M. Morari, "Certification aspects of the fast gradient method for solving the dual of parametric convex programs," *Math. Methods Oper. Res.*, vol. 77, no. 3, pp. 305–321, Jun. 2013.
[42] A. Bemporad and P. Patrinos, "Simple and certifiable quadratic programming algorithms for embedded linear model predictive control," in *Proc. 4th IFAC Conf. Nonlinear Model Predictive Control*, 2012, pp. 14–20.
[43] J. B. Rawlings, G. Pannocchia, S. J. Wright, and C. N. Bates, "On the inherent robustness of suboptimal model predictive control," in *Proc. SIAM Conf. Control Appl.*, San Diego, CA, USA, 2013, pp. 1–28.
[44] G. Frison, D. K. M. Kufualor, L. Imsland, and J. B. Jørgensen, "Efficient implementation of solvers for linear model predictive control on embedded devices," in *Proc. IEEE Multiconf. Syst. Control*, Nice, France, 2014.
[45] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, Jan. 2002.
[46] M. Herceg, M. Kvasnica, C. Jones, and M. Morari, "Multi-parametric toolbox 3.0," in *Proc. Eur. Control Conf.*, Zürich, Switzerland, 2013, pp. 502–510.
[47] A. Grancharova and T. A. Johansen, *Explicit Nonlinear Model Predictive Control*, vol. 429, *Lecture Notes in Control and Information Sciences*. Berlin, Germany: Springer-Verlag, 2012.
[48] T. Poggi, F. Comaschi, and M. Storace, "Digital circuit realization of piecewise affine functions with non-uniform resolution: Theory and FPGA implementation," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 57, no. 2, pp. 131–135, Feb. 2010.
[49] F. Bayat, T. A. Johansen, and A. A. Jalali, "Using hash tables to manage time-storage complexity in point location problem: Application to explicit MPC," *Automatica*, vol. 47, no. 3, pp. 571–577, Mar. 2011.
[50] F. Bayat, T. A. Johansen, and A. A. Jalali, "Flexible piecewise function evaluation methods based on truncated binary search trees and lattice representation in explicit MPC," *IEEE Trans. Control Syst. Technol.*, vol. 20, no. 3, pp. 632–640, May 2012.
[51] J. M. Maciejowski and C. N. Jones, "MPC fault-tolerant flight control case study: Flight 1862," in *Proc. IFAC Safeprocess Conf.*, 2003, pp. 121–126.
[52] P. Gawkowski, M. Lawrynczuk, P. Marusak, J. Sosnowski, and P. Tatjewski, "Fail-bounded implementations of the numerical model predictive control algorithms," *Control Cybern.*, vol. 39, no. 4, pp. 1117–1134, 2010.
[53] K. C. Ng, L. Wang, and I. D. Peake, "Safety-critical multi-core software architecture for model predictive control," in *Proc. Australian Control Conf.*, 2011, pp. 434–439.
[54] P. Gawkowski *et al.*, "Testing fault robustness of model predictive control algorithms," in *Architecting Critical Systems*, vol. 6150, *Lecture Notes in Computer Science*, H. Giese, Ed. Berlin, Germany: Springer-Verlag, 2010, pp. 109–124.
[55] N. G. Leveson, *Safeware. System Safety and Computers*, 1st ed. Reading, MA, USA: Addison-Wesley, 1995.
[56] D. Seto, B. Krogh, L. Sha, and A. Chutinan, "Dynamic control system upgrade using the simplex architecture," *IEEE Control Syst.*, vol. 18, no. 4, pp. 72–80, Aug. 1998.
[57] S. Bak *et al.*, "The system-level simplex architecture for improved real-time embedded system safety," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2009, pp. 99–107.
[58] R. Gu, S. S. Bhattacharyya, and W. S. Levine, "Methods for efficient implementation of model predictive control on multiprocessor systems," in *Proc. IEEE CCA*, 2010, pp. 1357–1362.
[59] S. Ruud and I. B. Utne, Verification and Examination Management by Marginal Verification Risk 2014, submitted for publication.
[60] P. Kapasouris, M. Athans, and G. Stein, "Design of feedback control systems for unstable plants with saturating actuators," in *Proc. IFAC Symp. Nonlin. Control Syst. Des.*, 1990, pp. 302–307.
[61] A. Bemporad, A. Casavola, and E. Mosca, "Nonlinear control of constrained linear systems via predictive reference management," *IEEE Trans. Autom. Control*, vol. 42, no. 3, pp. 340–349, Mar. 1997.

Tor A. Johansen He received the M.Sc. and Ph.D. degrees from Norwegian University of Science and Technology (NTNU), Trondheim, Norway.

He worked at SINTEF, Trondheim, Norway, as a Researcher before he was appointed Associated Professor with the NTNU, Trondheim, in 1997, and a Professor in 2001. He has published several hundred articles in the areas of control, estimation, and optimization with applications in the marine, automotive, biomedical, and process industries. In 2002, he cofounded the company Marine Cybernetics AS, where he was Vice President until 2008.

Prof. Johansen was the recipient of the 2006 Arch T. Colwell Merit Award of the SAE and is currently a Principal Researcher with the Center of Excellence on Autonomous Marine Operations and Systems and the Director of the Unmanned Aerial Vehicle Laboratory, NTNU.